Description

Verification of the availability of a server

5   The present invention relates to a method for verifying the availability of a server, a control program and a client for a network providing connectionless services.

Reliable verification of the availability of a server is of
10   great significance in particular in the case of loosely connected client-server relations in packet-oriented data networks. If a client detects early enough that a server is overloaded or down, then it is possible to implement countermeasures in good time, such as, for example, the search
15   for an alternative server or the creation of warning notices. Methods for verifying the availability of a server are applied, which methods also include keepalive tests described in the, H.323 standard, as on 11/2000, chapter 7.2.2, if there is no permanent communication relationship between client and server
20   and if, however, such a relationship must exist and be without fault is the basis for a desired functionality, for example, Internet telephony. In packet-oriented networks keepalive tests are used, for example, to simulate to the communication subscribers a quasi line-switching character with respect to
25   mutual availability.

With conventional keepalive tests, a client sends availability requests to a selected server in cyclical intervals of time. If the server transmits a response to the availability request
30   within a predefinable period of time, the server is deemed available and thus the communication relationship active. Of disadvantage in this approach is that the server is generally

put under a severe load by transmitting a response to all the
client requests.

5   The object underlying the invention is thus to specify a method
for verifying the availability of a server, which method
enables the load placed on the server by the availability
requests directed at it to be minimized, as well as to specify
technical implementations suitable for carrying out said
method.

10

This object is achieved according to the invention by a method
with the features specified in claim 1, a control program with
the features specified in claim 7 and a client with the
features given in claim 8. Advantageous developments of the

15   present invention are set down in the dependent claims.

An essential aspect of the present invention consists in that a
client, which has received a confirmation message from a server
in response to an availability request directed to said server,

20   transmits a message regarding the availability of the server to
other clients. Whereupon the predefinable other clients prevent
the transmission of an availability request to the server at
least for a predefinable period of time. In this way, it is
possible to considerably reduce the load placed on the server

25   by availability requests, without this being at the cost of,
for example, a server failure or a server overload no longer
being detected in good time.

The present invention is explained in greater detail below in

30   an exemplary embodiment with reference to a drawing, in which

Figure 1 shows an application environment of the present
invention with a large number of clients and servers
in a packet-oriented network with several subnetworks,

5 Figure 2 shows a flow chart for a method for verifying the
availability of a server,

Figure 3 a flow chart for, compared with Figure 2, a modified
method in which, in addition, the availability of
10   clients is checked by the server.

Figure 1 displays a communication system that comprises a
packet-oriented network with several subnetworks 101, 110, 120,
130, server 102 to 104 and several clients 111 to 115, 121 to
15 123, 131 to 133. The clients 111 to 115, 121 to 123, 131 to 133
use the services offered by the servers 102 to 104, for
example, Internet telephony.

The subnetworks 110, 120, 130 are local subnetworks, each of
20 which has a gatekeeper. The gatekeepers are used mainly for
access control and address translation services. Servers in the
packet-oriented network can assume this functionality of the
gatekeeper.

25 The clients 111 to 115, 121 to 123, 131 to 133 first register
with the respective server in order to use the services offered
by a server 102 to 104. Thereby, a period of time is specified
within which a registration is treated as valid. The sequence
of a registration process is represented in model form and
30 without any limitation of universal validity for client 111.

Client 111 has a central processing unit (CPU), a random access
memory (RAM), a hard drive for non-volatile storage of data

(HD), a network adaptor (Tx/Rx) and a timing unit (timer). The
remaining clients 112 to 115, 121 to 123, 131 to 133
represented in Figure 1 can also have such a structure and such
a functionality.

5

In order to register with one of the servers 102 to 104, the
client 111 transmits a message with a request to register RRQ
with a server that provides a desired service. In the message
with the request to register RRQ, there is a period of time
10   specified within which period the registration of client 111
with the server selected must be valid. Outside this period of
time, the registration of the client is considered invalid.

If it is available the selected server responds to the
15   registration request with a message containing registration
confirmation RCF. In the message with the registration
confirmation RCF, a period of validity is also specified for
the registration. The period of validity confirmed by the
selected server either corresponds to the period of time
20   specified by client 111 or is shorter than said period of time.

Within the period of validity for the registration, client 111
can direct availability requests to the selected server.
Availability requests of this type, which are also called
25   keepalive tests, are likewise transmitted to the selected
server in the form of a message with a registration request
RRQ. A message with a registration request RRQ sent in the
course of a keepalive test has, compared to the usual messages
with a registration request, a set keepalive bit. .

30

In accordance with the ITU-T recommendations H.225.0, as on
11/2000, chapter 7.9, a shortened message with a registration
request is adequate for a "lightweight registration". Thereby

the message with a registration request only has information
about the client's registration and status transport address
(RAS address), a set keepalive bit, an end point indicator for
the client, a server or gatekeeper identifier, tokens for
5 operations to be carried out and a period of validity for the
registration.

A message with request to register RRQ, which is transmitted to
the selected server during a keepalive tests, can also be used
10 to extend the period of validity of the registration. However,
this only applies if the message is received by the selected
server within the period of validity of the registration.
Therefore, messages und processing delays should be taken into
consideration when a client is notifying the end of a period of
15 validity. After the registration validity period has ended it
is no longer sufficient to transmit a shortened message with a
registration request as described above in order to renew the
registration.

20 If the message with the registration confirmation RCF does not
have any information as to the period of validity of the
registration, this will be interpreted as indicating that the
selected server does not support any keepalive mechanisms for
availability requests. If clients receive such a message
25 without information about a period of validity, they should
prevent availability requests being transmitted to the
respective server.

When the validity of a registration has expired, the selected
30 server can send a message to the client in question with a
corresponding indication in order to inform said client that
the period of validity has expired. This is of particular
advantage when there is a loss of synchronization between the

control units at the client on the one hand and at the server
on the other.

If, after the period of validity of the registration has ended,
5 client 111 transmits an above described shortened message with
registration request RRQ and set keepalive bit to the selected
server, then the server will respond with a message with
registration rejection RRJ. A reason for rejection can also be
included in such a message.
10

In order to avoid loading a server 102 to 104 unnecessarily
with availability requests from clients 111 to 115, 121 to 123,
131 to 133, a client that has received a positive or a negative
response to an availability request, uses a multicast message
15 MCM to inform all the clients within the same subnetwork 110,
120, 130 of the availability of the respective server. In
addition, a limitation on the same subnetwork allows problems
with respect to generally unsolvable addresses to be avoided.
In particular, unnecessary loading of the network with non-
20 addressable messages is prevented.

The flow chart represented in Figure 2 serves to further
illustrate a method for verifying the availability of a server
in a packet-oriented communications network. The starting point
25 of the method is the start of a timer or of a counter of a
timing unit of a client (step 201). Using the timer, times are
set at which the client's availability request are to be
transmitted to a selected server.

30 After the timer has started, the client monitors the reception
of the multicast messages MCM, which provide statements as to
the availability of a server (step 202). If the client receives
such a multicast message, then the timer is reset (step 208).

This thus prevents the transmission of availability requests to the selected server for a predefinable period of time.

Provided that no new multicast message was received, in
5   addition a check is made whether the timer has run out and thus a time arrived at for delivering a new availability request (step 203). If the timer has not yet run out, then the reception of multicast messages and the progress of the timer continues to be checked. If the timer has run out, an
10   availability request is sent to the selected server (step 204).

In step 205, the availability or unavailability of the selected server is determined at least implicitly. If the selected server is available, then said server responds to the
15   availability request by sending a confirmation message to the requesting client (step 206). After having received the confirmation message, the requesting client informs all other clients within the same subnetwork about the availability of the selected server by means of a multicast message (step 207).
20   The other clients who have received the multicast message, hereupon for their part prevent the transmission of availability requests to the selected server for a predefinable period of time.

25   As, when the selected server fails or is overloaded, no confirmation message regarding the availability of said server is sent, a check is made to determine whether the selected server is at least able to respond to the availability request with an unavailability message (step 209). If the selected
30   server were overloaded, this would, for example, still be possible. By contrast, if the selected server fails, one can no longer expect an unavailability message to be sent. Therefore, a check is made to determine whether a time limit for the

reception of a confirmation message or of an unavailability message has been reached (step 211). If this time limit is exceeded then the requesting client transmits a multicast message regarding the unavailability of the selected server to
5   all clients within the same subnetwork (step 210). The same procedure is followed if the requesting client receives an unavailability message within the time limit allowed for a response to the availability request.

10   The method for verifying the availability of a server described is preferably implemented by means of a control program which can be loaded in a random access memory of a client and has at least one code section, on the implementation of which control program, the steps described above proceed to run –
15   transmission of the availability request, monitoring of the receipt of a confirmation message, transmission of a message regarding the availability to other clients, monitoring the receipt of messages of other clients and if necessary preventing the sending of availability requests. By means of
20   the control program, a client for a communications network that provides connectionless services is implemented, which client has a device for transmitting availability requests, a device for monitoring the receipt of confirmation messages, a device for transmitting messages regarding the availability of servers
25   to other clients as well as a device for monitoring the receipt of messages of other clients and for preventing the transmission of availability requests.

The following observations serve to clarify the advantages of
30   the described method for checking the availability of a server compared with conventional keepalive tests used up to now. According to the conventional keepalive tests used up to now a server with $n=1000$ clients sending availability requests and a

request rate of a=3 requests per minute and client, and
assuming such requests were equally distributed in time, would
have to respond to an availability request every

5 $$[[t_r \ (a = 3, n = 1000) = \frac{60s}{a \cdot n} = \frac{60s}{3.1000} = 20ms]].$$

Thus, the server would be busy answering an availability
request in real time on average every 20 ms. This would result
in a considerable load being placed on the server by

10 availability requests, which would restrict it greatly in
realizing the tasks which it was actually intended for. The
first option to present itself for solving this problem would
be to make a large reduction in the clients' request rates per
minute, which would, however, result in the affected clients

15 possibly determining the loss of the availability of the server
far too late.

To assess one of the advantages that can be achieved with the
method proposed here, it is first assumed that n clients are

20 distributed on s subnetworks. It is further assumed that from
each of these subnetworks at least one client will try to query
the availability of the server. Each of these clients would
then inform the remaining clients within its subnetwork by
means of a multicast message regarding the availability of the

25 server. Under ideal conditions, the number of availability
requests to be processed by the server would correspond to the
number s of subnetworks. In addition, it must also be taken
into consideration that the multicast messages in the
subnetworks could have a loss rate v between 0 and 100 percent.

30 Thus the number v_c of clients per subnetwork, which are not
reached by a multicast message and hence themselves send an
availability request to the server depending on the loss rate v
of the multicast messages:

$$v_c = v(\frac{n}{s} - 1) .$$

The total number $v_c{}^{all}$ of all the clients who, under this
condition, send an availability request to the server, can thus
be estimated as follows:

$$v_c{}^{all} = s \cdot v \cdot (\frac{n}{s} - 1) + s = v \cdot n - v \cdot s + s = v \cdot n + s(1 - v).$$

Assuming the above, the period of time t_r between two
availability requests sent to the server can be calculated as
follows:

$$t_r (a,n,s,v) = \frac{60s}{a \cdot (v \cdot n + s(1 - v))}.$$

Using a numerical example, the load placed on the server by
availability requests in the conventional keepalive tests up to
now is compared with the load placed on the server in
accordance with the method proposed here. With a request rate
of a=3 availability requests per minute and n=1000 clients, the
period of time t_r between two availability requests is:

$$t_r(a=3,n=1000,s,v) = \frac{60s}{3 \cdot (v \cdot 1000 + s(1 - v))} = \frac{20s}{1000 \cdot v + s(1 - v)}.$$

Typical values for the loss rate v are clearly below 100
percent. With the above values for the request rate a, the
number n of the clients and s=10 subnetworks, produces, in
dependence on the loss rate v, the following efficiency
increases d_{eff} in percent in relation to the conventional

keepalive tests with a period of time t_r = 20 ms between two availability requests:

V	0.0	0.1	0.2	0.5	0.8
t_r [ms]	2000	183	96	40	25
d [%]	9900	815	380	100	25

5   With a loss rate v = 20 % the mean time t_r between two
    availability requests to be processed by the server is
    approximately 96 ms. This corresponds to an efficiency increase
    d_{eff} = 380 % compared with conventional keepalive tests up to
    now.

10

    All in all, one can determine that, with the method proposed
    for checking the availability of the server, the load placed on
    the server by responding to availability requests clearly
    lessens, without this resulting in a significant increase in
15 the time until the unavailability of the server is known to the
    clients. Moreover, in order to implement the method proposed
    here, changes are only necessary on the client side. The
    previous procedure for processing availability requests can be
    kept on the server side and thus does not require any further
20 expenditure.

    In the method represented in Figure 3, a server also checks the
    availability of clients. This means that a server is also
    informed about the availability of clients who are only
25 informed about the availability of the server by another client
    per multicast message and who do not themselves send keepalive
    requests directly to the server. First each client starts a
    first timer (timer 1) (step 301) and checks whether a multicast
    collective request has been received (step 302). A multicast
30 collective request can be sent by a client as a sign that this

client would like to carry out a collective keepalive test with
a server, and that this client needs the keepalive data of
other clients for this.

5   The status of the first timer is checked continuously by the
respective client (step 303). If the first timer has not yet
run out and no multicast collective request has been received,
then the receipt of a multicast collective request continues to
be monitored. If the first timer has run out and no multicast
10   collective request has been received, then the respective
client takes on the task of sending a multicast collective
request to a selected server itself. To this end the respective
requesting client starts a second timer (timer 2) (step 304)
and transmits a multicast collective request to predefinable
15   other clients (step 305). Subsequently, the respective
requesting client checks whether the multicast collective
request was responded to by the predefinable other clients
(step 306), and whether the second timer has run out (step
307). If the second timer has not yet run out and no multicast
20   collective request has been received, then the receipt of a
response to the multicast collective request continues to be
monitored.

After the second timer has run out, the respective requesting
25   client starts a third timer (timer 3) (step 308) and sends a
collective availability request to the selected server (step
309). The collective availability request includes data of the
respective requesting client and of those predefinable other
clients who responded to the multicast collective request of
30   the respective requesting client until the second timer ran
out. Subsequently the respective requesting client checks
whether the selected server has signaled availability (step
310), and whether the third timer has run out (step 311). If

the third timer has not yet run out and no response has been
received to the collective availability request, then the
receipt of a response to the collective availability request
continues to be monitored.

5

If the selected server has signaled its availability to the
respective requesting client, then the respective requesting
client transmits a positive multicast availability message to
the predefinable further clients (step 312). If the selected
10 server has signaled its unavailability or has not sent any
response to the collective availability request by the time the
third timer has run out, then the respective requesting client
transmits a negative multicast availability message to the
predefinable other clients (step 313).

15

Predefinable other clients who have received a multicast
collective request in accordance with the check in step 302,
transmit their keepalive data to the respective requesting
client (step 314), the receipt of which data the respective
20 requesting client checks in accordance with 306. Subsequently,
the predefinable other clients start a fourth timer (timer 4)
(step 315) and check whether a positive or negative multicast
availability message of the respective requesting client was
received (step 316), and whether the fourth timer has run out
25 (step 317). On receipt of a multicast availability message of
the respective requesting client, the first timer is reset
(step 318), so that the predefinable other clients do not send
any availability requests to the selected server for a period
of time determined by the first timer running out.

30

As long as the fourth timer has not run out, the predefinable
other clients wait for a multicast availability message of the
respective requesting client. If this does not occur before the

fourth timer runs out, the respective client then itself
assumes a role as the placer of a collective availability
request in accordance with steps 304 to 313, provided the first
timer has run out or no further multicast collective request
5   has been received before said first timer has run out.

The application of the present invention is not limited to the
exemplary embodiment described here.